Noise2Noise: Learning Image Restoration without Clean Data

Jaakko Lehtinen[1 2] **Jacob Munkberg**[1] **Jon Hasselgren**[1] **Samuli Laine**[1] **Tero Karras**[1] **Miika Aittala**[3] **Timo Aila**[1]

Abstract

We apply basic statistical reasoning to signal reconstruction by machine learning – learning to map corrupted observations to clean signals – with a simple and powerful conclusion: it is possible to learn to restore images by only looking at corrupted examples, at performance at and sometimes exceeding training using clean data, without explicit image priors or likelihood models of the corruption. In practice, we show that a single model learns photographic noise removal, denoising synthetic Monte Carlo images, and reconstruction of undersampled MRI scans – all corrupted by different processes – based on noisy data only.

1. Introduction

Signal reconstruction from corrupted or incomplete measurements is an important subfield of statistical data analysis. Recent advances in deep neural networks have sparked significant interest in avoiding the traditional, explicit a priori statistical modeling of signal corruptions, and instead *learning* to map corrupted observations to the unobserved clean versions. This happens by training a regression model, e.g., a convolutional neural network (CNN), with a large number of pairs $(\hat{x}_i, y_i)$ of corrupted inputs $\hat{x}_i$ and clean targets y_i and minimizing the empirical risk

$$\underset{\theta}{\operatorname{argmin}} \sum_i L\left(f_\theta(\hat{x}_i),\, y_i\right), \tag{1}$$

where f_θ is a parametric family of mappings (e.g., CNNs), under the loss function L. We use the notation $\hat{x}$ to underline the fact that the corrupted input $\hat{x} \sim p(\hat{x}|y_i)$ is a random variable distributed according to the clean target. Training data may include, for example, pairs of short and long exposure photographs of the same scene, incomplete and complete k-space samplings of magnetic resonance images, fast-but-noisy and slow-but-converged ray-traced renderings of a synthetic scene, etc. Significant advances have been reported in several applications, including Gaussian denoising, de-JPEG, text removal (Mao et al., 2016), super-resolution (Ledig et al., 2017), colorization (Zhang et al., 2016), and image inpainting (Iizuka et al., 2017). Yet, obtaining clean training targets is often difficult or tedious: a noise-free photograph requires a long exposure; full MRI sampling precludes dynamic subjects; etc.

In this work, we observe that we can often *learn to turn bad images into good images by only looking at bad images*, and do this just as well – sometimes even better – as if we were using clean examples. Further, we require neither an explicit statistical likelihood model of the corruption nor an image prior, and instead learn these indirectly from the training data. (Indeed, in one of our examples, synthetic Monte Carlo renderings, the non-stationary noise cannot be characterized analytically.) In addition to denoising, our observation is directly applicable to inverse problems such as MRI reconstruction from undersampled data. While our conclusion is almost trivial from a statistical perspective, it significantly eases practical learned signal reconstruction by lifting requirements on availability of training data.

The reference TensorFlow implementation for Noise2Noise training is available on GitHub.[1]

2. Theoretical Background

Assume that we have a set of unreliable measurements $(y_1, y_2, ...)$ of the room temperature. A common strategy for estimating the true unknown temperature is to find a number z that has the smallest average deviation from the measurements according to some loss function L:

$$\underset{z}{\operatorname{argmin}}\, \mathbb{E}_y\{L(z,\, y)\}. \tag{2}$$

For the L_2 loss $L(z, y) = (z - y)^2$, this minimum is found at the arithmetic mean of the observations:

$$z = \mathbb{E}_y\{y\}. \tag{3}$$

The L_1 loss, the sum of absolute deviations $L(z, y) = |z - y|$, in turn, has its optimum at the median of the observations. The general class of deviation-minimizing estimators are

[1]NVIDIA [2]Aalto University [3]MIT CSAIL. Correspondence to: Jaakko Lehtinen <jlehtinen@nvidia.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

[1] https://github.com/NVlabs/noise2noise

known as M-estimators (Huber, 1964). From a statistical viewpoint, summary estimation using these common loss functions can be seen as ML estimation by interpreting the loss function as the negative log likelihood.

Training neural network regressors is a generalization of this point estimation procedure. Observe the form of the typical training task for a set of input-target pairs (x_i, y_i), where the network function $f_\theta(x)$ is parameterized by θ:

$$\underset{\theta}{\operatorname{argmin}} \mathbb{E}_{(x,y)}\{L(f_\theta(x),\, y)\}. \tag{4}$$

Indeed, if we remove the dependency on input data, and use a trivial f_θ that merely outputs a learned scalar, the task reduces to (2). Conversely, the full training task decomposes to the same minimization problem at every training sample; simple manipulations show that (4) is equivalent to

$$\underset{\theta}{\operatorname{argmin}} \mathbb{E}_x\{\mathbb{E}_{y|x}\{L(f_\theta(x),\, y)\}\}. \tag{5}$$

The network can, in theory, minimize this loss by solving the point estimation problem separately for each input sample. Hence, the properties of the underlying loss are inherited by neural network training.

The usual process of training regressors by Equation 1 over a finite number of input-target pairs (x_i, y_i) hides a subtle point: instead of the 1:1 mapping between inputs and targets (falsely) implied by that process, in reality the mapping is multiple-valued. For example, in a superresolution task (Ledig et al., 2017) over all natural images, a low-resolution image x can be explained by many different high-resolution images y, as knowledge about the exact positions and orientations of the edges and texture is lost in decimation. In other words, $p(y|x)$ is the highly complex distribution of natural images consistent with the low-resolution x. Training a neural network regressor using training pairs of low- and high-resolution images using the L_2 loss, the network learns to output the average of all plausible explanations (e.g., edges shifted by different amounts), which results in spatial blurriness for the network's predictions. A significant amount of work has been done to combat this well known tendency, for example by using learned discriminator functions as losses (Ledig et al., 2017; Isola et al., 2017).

Our observation is that for certain problems this tendency has an unexpected benefit. A trivial, and, at first sight, useless, property of L_2 minimization is that on expectation, the estimate remains unchanged if we replace the targets with random numbers whose expectations match the targets. This is easy to see: Equation (3) holds, no matter what particular distribution the ys are drawn from. Consequently, the optimal network parameters θ of Equation (5) also remain unchanged, if input-conditioned target distributions $p(y|x)$ are replaced with arbitrary distributions that have the same conditional expected values. *This implies that we can, in principle, corrupt the training targets of a neural network with zero-mean noise without changing what the network learns.* Combining this with the corrupted inputs from Equation 1, we are left with the empirical risk minimization task

$$\underset{\theta}{\operatorname{argmin}} \sum_i L\left(f_\theta(\hat{x}_i),\, \hat{y}_i\right), \tag{6}$$

where both the inputs and the targets are now drawn from a corrupted distribution (not necessarily the same), conditioned on the underlying, unobserved clean target y_i such that $\mathbb{E}\{\hat{y}_i|\hat{x}_i\} = y_i$. Given infinite data, the solution is the same as that of (1). For finite data, the variance is the average variance of the corruptions in the targets, divided by the number of training samples (see appendix). Interestingly, none of the above relies on a likelihood model of the corruption, nor a density model (prior) for the underlying clean image manifold. That is, we do not need an explicit $p(\text{noisy}|\text{clean})$ or $p(\text{clean})$, as long as we have data distributed according to them.

In many image restoration tasks, the expectation of the corrupted input data *is* the clean target that we seek to restore. Low-light photography is an example: a long, noise-free exposure is the average of short, independent, noisy exposures. With this in mind, the above suggests the ability to learn to remove photon noise given only pairs of noisy images, with no need for potentially expensive or difficult long exposures. Similar observations can be made about other loss functions. For instance, the L_1 loss recovers the median of the targets, meaning that neural networks can be trained to repair images with significant (up top 50%) outlier content, again only requiring access to pairs of such corrupted images.

In the next sections, we present a wide variety of examples demonstrating that these theoretical capabilities are also efficiently realizable in practice.

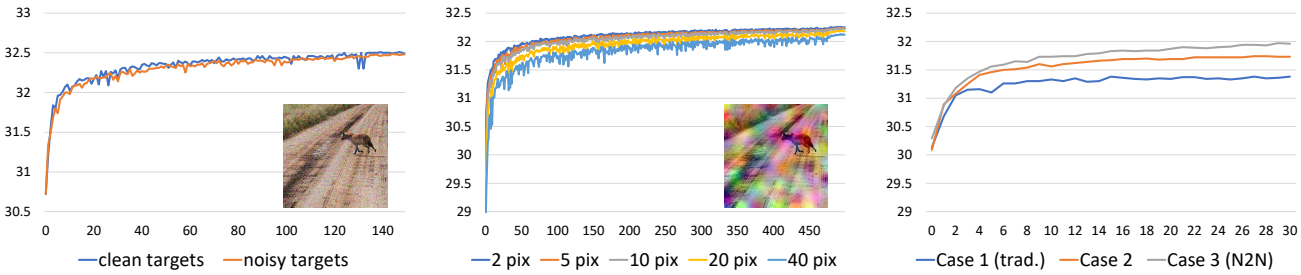
3. Practical Experiments

We now experimentally study the practical properties of noisy-target training. We start with simple noise distributions (Gaussian, Poisson, Bernoulli) in Sections 3.1 and 3.2, and continue to the much harder, analytically intractable Monte Carlo image synthesis noise (Section 3.3). In Section 3.4, we show that image reconstruction from sub-Nyquist spectral samplings in magnetic resonance imaging (MRI) can be learned from corrupted observations only.

3.1. Additive Gaussian Noise

We will first study the effect of corrupted targets using synthetic additive Gaussian noise. As the noise has zero mean, we use the L_2 loss for training to recover the mean.

Our baseline is a recent state-of-the-art method "RED30" (Mao et al., 2016), a 30-layer hierarchical residual net-



(a) White Gaussian, $\sigma = 25$ **(b) Brown Gaussian, $\sigma = 25$** **(c) Capture budget study (see text)**

Figure 1. Denoising performance (dB in KODAK dataset) as a function of training epoch for additive Gaussian noise. (a) For i.i.d. (white) Gaussian noise, clean and noisy targets lead to very similar convergence speed and eventual quality. (b) For brown Gaussian noise, we observe that increased inter-pixel noise correlation (wider spatial blur; one graph per bandwidth) slows convergence down, but eventual performance remains close. (c) Effect of different allocations of a fixed capture budget to noisy vs. clean examples (see text).

Table 1. PSNR results from three test datasets KODAK, BSD300, and SET14 for Gaussian, Poisson, and Bernoulli noise. The comparison methods are BM3D, Inverse Anscombe transform (ANSC), and deep image prior (DIP).

	Gaussian (σ=25)			Poisson (λ=30)			Bernoulli (p=0.5)		
	clean	noisy	BM3D	clean	noisy	ANSC	clean	noisy	DIP
Kodak	32.50	32.48	31.82	31.52	31.50	29.15	33.01	33.17	30.78
BSD300	31.07	31.06	30.34	30.18	30.16	27.56	31.04	31.16	28.97
Set14	31.31	31.28	30.50	30.07	30.06	28.36	31.51	31.72	30.67
Average	**31.63**	**31.61**	**30.89**	**30.59**	**30.57**	**28.36**	**31.85**	**32.02**	**30.14**

work with 128 feature maps, which has been demonstrated to be very effective in a wide range of image restoration tasks, including Gaussian noise. We train the network using 256×256-pixel crops drawn from the 50k images in the IMAGENET validation set. We furthermore randomize the noise standard deviation $\sigma \in [0, 50]$ separately for each training example, i.e., the network has to estimate the magnitude of noise while removing it ("blind" denoising).

We use three well-known datasets: BSD300 (Martin et al., 2001), SET14 (Zeyde et al., 2010), and KODAK[2]. As summarized in Table 1, the behavior is qualitatively similar in all three sets, and thus we discuss the averages. When trained using the standard way with clean targets (Equation 1), RED30 achieves 31.63 ± 0.02 dB with $\sigma = 25$. The confidence interval was computed by sampling five random initializations. The widely used benchmark denoiser BM3D (Dabov et al., 2007) gives ∼0.7 dB worse results. When we modify the training to use noisy targets (Equation 6) instead, the denoising performance remains equally good. Furthermore, the training converges just as quickly, as shown in Figure 1a. This leads us to conclude that clean targets are unnecessary in this application. This perhaps surprising observation holds also with different networks and network capacities. Figure 2a shows an example result.

[2]http://r0k.us/graphics/kodak/

For all further tests, we switch from RED30 to a shallower U-Net (Ronneberger et al., 2015) that is roughly 10× faster to train and gives similar results (−0.2 dB in Gaussian noise). The architecture and training parameters are described in the appendix.

Convergence speed Clearly, every training example asks for the impossible: there is no way the network could succeed in transforming one instance of the noise to another. Consequently, the training loss does actually not decrease during training, and the loss gradients continue to be quite large. Why do the larger, noisier gradients not affect convergence speed? While the activation gradients are indeed noisy, the weight gradients are in fact relatively clean because Gaussian noise is independent and identically distributed (i.i.d.) in all pixels, and the weight gradients get averaged over 2^{16} pixels in our fully convolutional network.

Figure 1b makes the situation harder by introducing inter-pixel correlation to the noise. This brown additive noise is obtained by blurring white Gaussian noise by a spatial Gaussian filter of different bandwidths and scaling to retain $\sigma = 25$. An example is shown in Figure 1b. As the correlation increases, the effective averaging of weight gradients decreases, and the weight updates become noisier. This makes the convergence slower, but even with extreme blur, the eventual quality is similar (within 0.1 dB).

Finite data and capture budget The previous studies relied on the availability of infinitely many noisy examples produced by adding synthetic noise to clean images. We now study corrupted vs. clean training data in the realistic scenario of finite data and a fixed capture budget. Our experiment setup is as follows. Let one ImageNet image with white additive Gaussian noise at $\sigma = 25$ correspond to one "capture unit" (CU). Suppose that 19 CUs are enough for a clean capture, so that one noisy realization plus the clean version (the average of 19 noisy realizations) consumes 20 CU. Let us fix a total capture budget of, say, 2000 CUs. This budget can be allocated between clean latents

(N) and noise realizations per clean latent (M) such that $N * M = 2000$. In the traditional scenario, we have only 100 training pairs ($N = 100$, $M = 20$): a single noisy realization and the corresponding clean image (= average of 19 noisy images; Figure 1c, Case 1). We first observe that using the *same* captured data as $100 * 20 * 19 = 38000$ training pairs with corrupted targets — i.e., for each latent, forming all the $19 * 20$ possible noisy/clean pairs — yields notably better results (several .1s of dB) than the traditional, fixed noisy+clean pairs, even if we still only have $N = 100$ latents (Figure 1c, Case 2). Second, we observe that setting $N = 1000$ and $M = 2$, i.e., increasing the number of clean latents but only obtaining two noisy realizations of each (resulting in 2000 training pairs) yields even better results (again, by several .1s of dB, Figure 1c, Case 3).

We conclude that for additive Gaussian noise, *corrupted targets offer benefits — not just the same performance but better — over clean targets* on two levels: both 1) seeing more realizations of the corruption for the same latent clean image, and 2) seeing more latent clean images, even if just two corrupted realizations of each, are beneficial.

3.2. Other Synthetic Noises

We will now experiment with other types of synthetic noise. The training setup is the same as described above.

Poisson noise is the dominant source of noise in photographs. While zero-mean, it is harder to remove because it is signal-dependent. We use the L_2 loss, and vary the noise magnitude $\lambda \in [0, 50]$ during training. Training with clean targets results in 30.59 ± 0.02 dB, while noisy targets give an equally good 30.57 ± 0.02 dB, again at similar convergence speed. A comparison method (Mäkitalo & Foi, 2011) that first transforms the input Poisson noise into Gaussian (Anscombe transform), then denoises by BM3D, and finally inverts the transform, yields 2 dB less.

Other effects, e.g., dark current and quantization, are dominated by Poisson noise, can be made zero-mean (Hasinoff et al., 2016), and hence pose no problems for training with noisy targets. We conclude that noise-free training data is unnecessary in this application. That said, saturation (gamut clipping) renders the expectation incorrect due to removing part of the distribution. As saturation is unwanted for other reasons too, this is not a significant limitation.

Multiplicative Bernoulli noise (aka binomial noise) constructs a random mask m that is 1 for valid pixels and 0 for zeroed/missing pixels. To avoid backpropagating gradients from missing pixels, we exclude them from the loss:

$$\underset{\theta}{\operatorname{argmin}} \sum_i (m \odot (f_\theta(\hat{x}_i) - \hat{y}_i))^2, \qquad (7)$$

as described by Ulyanov et al. (2017) in the context of their deep image prior (DIP).

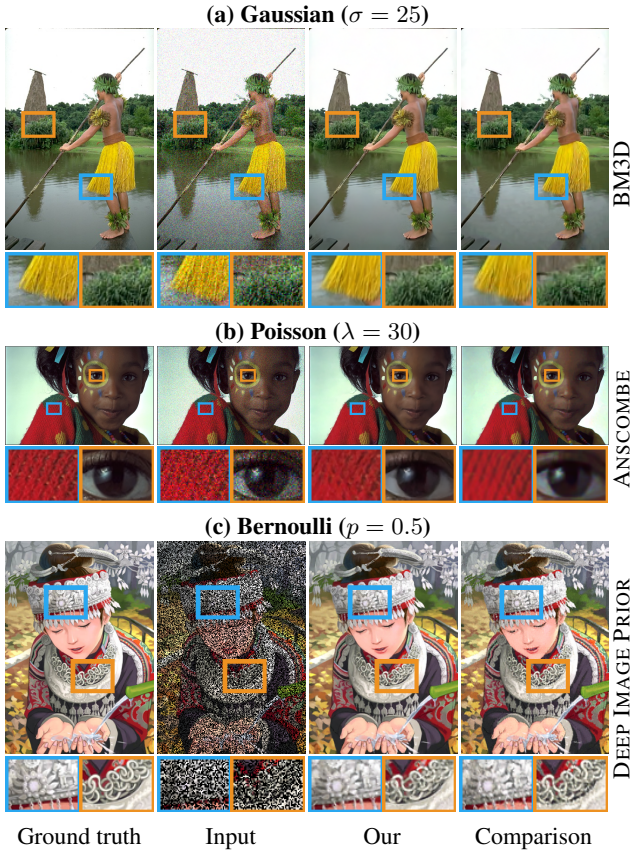


Figure 2. Example results for Gaussian, Poisson, and Bernoulli noise. Our result was computed by using noisy targets — the corresponding result with clean targets is omitted because it is virtually identical in all three cases, as discussed in the text. A different comparison method is used for each noise type.

The probability of corrupted pixels is denoted with p; in our training we vary $p \in [0.0, 0.95]$ and during testing $p = 0.5$. Training with clean targets gives an average of 31.85 ± 0.03 dB, noisy targets (separate m for input and target) give a slightly higher 32.02 ± 0.03 dB, possibly because noisy targets effectively implement a form of dropout (Srivastava et al., 2014) at the network output. DIP was almost 2 dB worse – DIP is not a learning-based solution, and as such very different from our approach, but it shares the property that neither clean examples nor an explicit model of the corruption is needed. We used the "Image reconstruction" setup as described in the DIP supplemental material.[3]

Text removal Figure 3 demonstrates blind text removal. The corruption consists of a large, varying number of random strings in random places, also on top of each other, and furthermore so that the font size and color are randomized as well. The font and string orientation remain fixed.

The network is trained using independently corrupted input

[3] https://dmitryulyanov.github.io/deep_image_prior

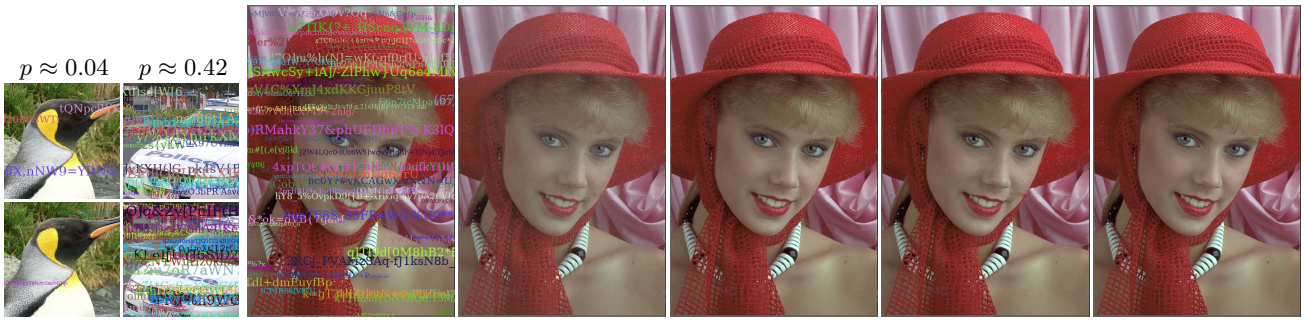


Figure 3. Removing random text overlays corresponds to seeking the median pixel color, accomplished using the L_1 loss. The mean (L_2 loss) is not the correct answer: note shift towards mean text color. Only corrupted images shown during training.

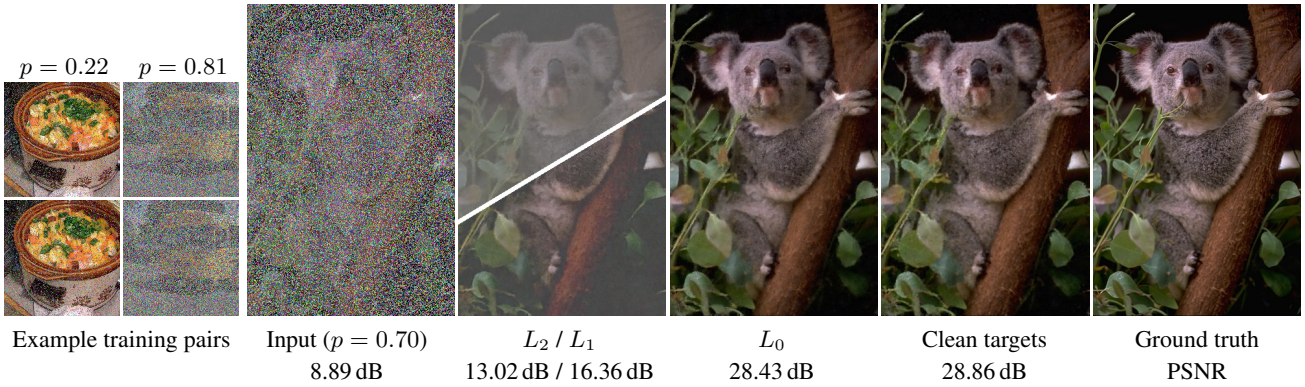


Figure 4. For random impulse noise, the approx. mode-seeking L_0 loss performs better than the mean (L_2) or median (L_1) seeking losses.

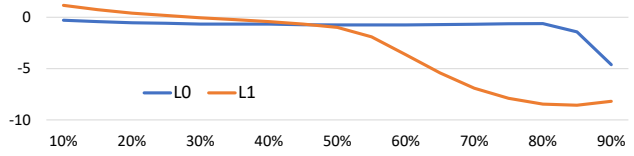


Figure 5. PSNR of noisy-target training relative to clean targets with a varying percentage of target pixels corrupted by RGB impulse noise. In this test a separate network was trained for each corruption level, and the graph was averaged over the KODAK dataset.

and target pairs. The probability of corrupted pixels p is approximately $[0, 0.5]$ during training, and $p \approx 0.25$ during testing. In this test the mean (L_2 loss) is not the correct answer because the overlaid text has colors unrelated to the actual image, and the resulting image would incorrectly tend towards a linear combination of the right answer and the average text color (medium gray). However, with any reasonable amount of overlaid text, a pixel retains the original color more often than not, and therefore the median is the correct statistic. Hence, we use $L_1 = |f_\theta(\hat{x}) - \hat{y}|$ as the loss function. Figure 3 shows an example result.

Random-valued impulse noise replaces some pixels with noise and retains the colors of others. Instead of the standard salt and pepper noise (randomly replacing pixels with black or white), we study a harder distribution where each pixel is replaced with a random color drawn from the uniform distribution $[0, 1]^3$ with probability p and retains its color with probability $1 - p$. The pixels' color distributions are a Dirac at the original color plus a uniform distribution, with relative weights given by the replacement probability p. In this case, neither the mean nor the median yield the correct result; the desired output is the *mode* of the distribution (the Dirac spike). The distribution remains unimodal. For approximate mode seeking, we use an annealed version of the "L_0 loss" function defined as $(|f_\theta(\hat{x}) - \hat{y}| + \epsilon)^\gamma$, where $\epsilon = 10^{-8}$, where γ is annealed linearly from 2 to 0 during training. This annealing did not cause any numerical issues in our tests. The relationship of the L_0 loss and mode seeking is analyzed in the appendix.

We again train the network using noisy inputs and noisy targets, where the probability of corrupted pixels is randomized separately for each pair from $[0, 0.95]$. Figure 4 shows the inference results when 70% input pixels are randomized.

Training with L_2 loss biases the results heavily towards gray, because the result tends towards a linear combination the correct answer and and mean of the uniform random corruption. As predicted by theory, the L_1 loss gives good results as long as fewer than 50% of the pixels are randomized, but beyond that threshold it quickly starts to bias dark and bright areas towards gray (Figure 5). L_0, on the other hand, shows little bias even with extreme corruptions (e.g. 90% pixels), because of all the possible pixel values, the correct answer (e.g. 10%) is still the most common.

3.3. Monte Carlo Rendering

Physically accurate renderings of virtual environments are most often generated through a process known as Monte Carlo path tracing. This amounts to drawing random sequences of scattering events ("light paths") in the scene that connect light sources and virtual sensors, and integrating the radiance carried by them over all possible paths (Veach & Guibas, 1995). The Monte Carlo integrator is constructed such that the intensity of each pixel is the expectation of the random path sampling process, i.e., the sampling noise is zero-mean. However, despite decades of research into importance sampling techniques, little else can be said about the distribution. It varies from pixel to pixel, heavily depends on the scene configuration and rendering parameters, and can be arbitrarily multimodal. Some lighting effects, such as focused caustics, also result in extremely long-tailed distributions with rare, bright outliers.

All of these effects make the removal of Monte Carlo noise much more difficult than removing, e.g., Gaussian noise. On the other hand, the problem is somewhat alleviated by the possibility of generating auxiliary information that has been empirically found to correlate with the clean result during data generation. In our experiments, the denoiser input consists of not only the per-pixel luminance values, but also the average albedo (i.e., texture color) and normal vector of the surfaces visible at each pixel.

High dynamic range (HDR) Even with adequate sampling, the floating-point pixel luminances may differ from each other by several orders of magnitude. In order to construct an image suitable for the generally 8-bit display devices, this high dynamic range needs to be compressed to a fixed range using a tone mapping operator (Cerdá-Company et al., 2016). We use a variant of Reinhard's global operator (Reinhard et al., 2002): $T(v) = (v/(1+v))^{1/2.2}$, where v is a scalar luminance value, possibly pre-scaled with an image-wide exposure constant. This operator maps any $v \geq 0$ into range $0 \leq T(v) < 1$.

The combination of virtually unbounded range of luminances and the nonlinearity of operator T poses a problem. If we attempt to train a denoiser that outputs luminance values v, a standard MSE loss $L_2 = (f_\theta(\hat{x}) - \hat{y})^2$ will be dominated by the long-tail effects (outliers) in the targets, and training does not converge. On the other hand, if the denoiser were to output tonemapped values $T(v)$, the nonlinearity of T would make the expected value of noisy target images $\mathbb{E}\{T(v)\}$ different from the clean training target $T(\mathbb{E}\{v\})$, leading to incorrect predictions.

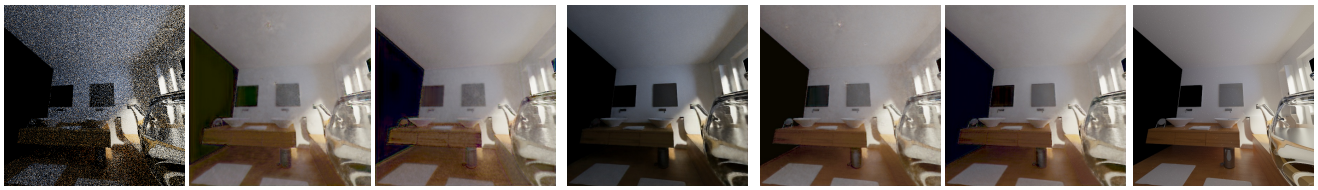
A metric often used for measuring the quality of HDR images is the relative MSE (Rousselle et al., 2011), where the squared difference is divided by the square of approximate luminance of the pixel, i.e., $(f_\theta(\hat{x}) - \hat{y})^2/(\hat{y} + \epsilon)^2$. However, this metric suffers from the same nonlinearity problem as comparing of tonemapped outputs. Therefore, we propose to use the network output, which tends towards the correct value in the limit, in the denominator: $L_{\text{HDR}} = (f_\theta(\hat{x}) - \hat{y})^2/(f_\theta(\hat{x}) + 0.01)^2$. It can be shown that L_{HDR} converges to the correct expected value as long as we consider the gradient of the denominator to be zero.

Finally, we have observed that it is beneficial to tone map the input image $T(\hat{x})$ instead of using HDR inputs. The network continues to output non-tonemapped (linear-scale) luminance values, retaining the correctness of the expected value. Figure 6 evaluates the different loss functions.

Denoising Monte Carlo rendered images We trained a denoiser for Monte Carlo path traced images rendered using 64 samples per pixel (spp). Our training set consisted of 860 architectural images, and the validation was done using 34 images from a different set of scenes. Three versions of the training images were rendered: two with 64 spp using different random seeds (noisy input, noisy target), and one with 131k spp (clean target). The validation images were rendered in both 64 spp (input) and 131k spp (reference) versions. All images were 960×540 pixels in size, and as mentioned earlier, we also saved the albedo and normal buffers for all of the input images. Even with such a small dataset, rendering the 131k spp clean images was a strenuous effort — for example, Figure 7d took 40 minutes to render on a high-end graphics server with 8 × NVIDIA Tesla P100 GPUs and a 40-core Intel Xeon CPU.

The average PSNR of the 64 spp validation inputs with respect to the corresponding reference images was 22.31 dB (see Figure 7a for an example). The network trained for 2000 epochs using clean target images reached an average PSNR of 31.83 dB on the validation set, whereas the similarly trained network using noisy target images gave 0.5 dB less. Examples are shown in Figure 7b,c – the training took 12 hours with a single NVIDIA Tesla P100 GPU.

At 4000 epochs, the noisy targets matched 31.83 dB, i.e., noisy targets took approximately twice as long to converge. However, the gap between the two methods had not narrowed appreciably, leading us to believe that some quality difference will remain even in the limit. This is not sur-



Input, 8 spp	L_2 with $\hat{x}, \hat{y}$	L_2 with $T(\hat{x}), \hat{y}$	L_2 with $T(\hat{x}), T(\hat{y})$	L_{HDR} with $\hat{x}, \hat{y}$	L_{HDR} with $T(\hat{x}), \hat{y}$	Reference, 32k spp
11.32 dB	25.46 dB	25.39 dB	15.50 dB	29.05 dB	30.09 dB	PSNR

Figure 6. Comparison of various loss functions for training a Monte Carlo denoiser with noisy target images rendered at 8 samples per pixel (spp). In this high-dynamic range setting, our custom relative loss L_{HDR} is clearly superior to L_2. Applying a non-linear tone map to the inputs is beneficial, while applying it to the target images skews the distribution of noise and leads to wrong, visibly too dark results.

(a) Input (64 spp), 23.93 dB (b) Noisy targets, 32.42 dB (c) Clean targets, 32.95 dB (d) Reference (131k spp)

Figure 7. Denoising a Monte Carlo rendered image. (a) Image rendered with 64 samples per pixel. (b) Denoised 64 spp input, trained using 64 spp targets. (c) Same as previous, but trained on clean targets. (d) Reference image rendered with 131 072 samples per pixel. PSNR values refer to the images shown here, see text for averages over the entire validation set.

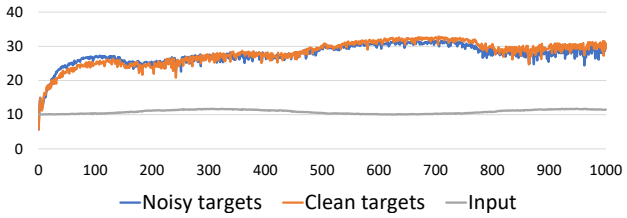


Figure 8. Online training PSNR during a 1000-frame flythrough of the scene in Figure 6. Noisy target images are almost as good for learning as clean targets, but are over 2000× faster to render (190 milliseconds vs 7 minutes per frame in this scene). Both denoisers offer a substantial improvement over the noisy input.

prising, since the training dataset contained only a limited number of training pairs (and thus noise realizations) due to the cost of generating the clean target images, and we wanted to test both methods using matching data. That said, given that noisy targets are 2000 times faster to produce, one could trivially produce a larger quantity of them and still realize vast gains. The finite capture budget study (Section 3.1) supports this hypothesis.

Online training Since it can be tedious to collect a sufficiently large corpus of Monte Carlo images for training a generally applicable denoiser, a possibility is to train a model specific to a single 3D scene, e.g., a game level or a movie shot (Chaitanya et al., 2017). In this context, it can even be desirable to train on-the-fly while walking through the scene. In order to maintain interactive frame rates, we can afford only few samples per pixel, and thus both input and target images will be inherently noisy.

Figure 8 shows the convergence plots for an experiment where we trained a denoiser from scratch for the duration of 1000 frames in a scene flythrough. On an NVIDIA Titan V GPU, path tracing a single 512×512 pixel image with 8 spp took 190 ms, and we rendered two images to act as input and target. A single network training iteration with a random 256×256 pixel crop took 11.25 ms and we performed eight of them per frame. Finally, we denoised both rendered images, each taking 15 ms, and averaged the result to produce the final image shown to the user. Rendering, training and inference took 500 ms/frame.

Figure 8 shows that training with clean targets does not perform appreciably better than noisy targets. As rendering a single clean image takes approx. 7 minutes in this scene (resp. 190 ms for a noisy target), the quality/time tradeoff clearly favors noisy targets.

3.4. Magnetic Resonance Imaging (MRI)

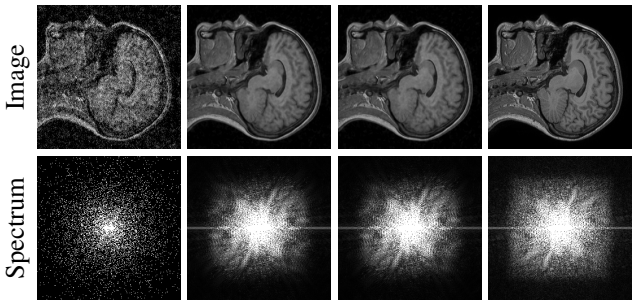
Magnetic Resonance Imaging (MRI) produces volumetric images of biological tissues essentially by sampling the

Fourier transform (the "k-space") of the signal. Modern MRI techniques have long relied on compressed sensing (CS) to cheat the Nyquist-Shannon limit: they undersample k-space, and perform non-linear reconstruction that removes aliasing by exploiting the sparsity of the image in a suitable transform domain (Lustig et al., 2008).

We observe that if we turn the k-space sampling into a random process with a known probability density $p(k)$ over the frequencies k, our main idea applies. In particular, we model the k-space sampling operation as a Bernoulli process where each individual frequency has a probability $p(k) = e^{-\lambda|k|}$ of being selected for acquisition.[4] The frequencies that are retained are weighted by the inverse of the selection probability, and non-chosen frequencies are set to zero. Clearly, the expectation of this "Russian roulette" process is the correct spectrum. The parameter λ controls the overall fraction of k-space retained; in the following experiments, we choose it so that 10% of the samples are retained relative to a full Nyquist-Shannon sampling. The undersampled spectra are transformed to the primal image domain by the standard inverse Fourier transform. An example of an undersampled input/target picture, the corresponding fully sampled reference, and their spectra, are shown in Figure 9(a, d).

Now we simply set up a regression problem of the form (6) and train a convolutional neural network using pairs of two independent undersampled images $\hat{x}$ and $\hat{y}$ of the same volume. As the spectra of the input and target are correct on expectation, and the Fourier transform is linear, we use the L_2 loss. Additionally, we improve the result slightly by enforcing the exact preservation of frequencies that are present in the input image $\hat{x}$ by Fourier transforming the result $f_\theta(\hat{x})$, replacing the frequencies with those from the input, and transforming back to the primal domain before computing the loss: the final loss reads $(\mathcal{F}^{-1}(R_{\hat{x}}(\mathcal{F}(f_\theta(\hat{x})))) - \hat{y})^2$, where R denotes the replacement of non-zero frequencies from the input. This process is trained end-to-end.

We perform experiments on 2D slices extracted from the IXI brain scan MRI dataset.[5] To simulate spectral sampling, we draw random samples from the FFT of the (already reconstructed) images in the dataset. Hence, in deviation from actual MRI samples, our data is real-valued and has the periodicity of the discrete FFT built-in. The training set contained 5000 images in 256×256 resolution from 50 subjects, and for validation we chose 1000 random images from 10 different subjects. The baseline PSNR of the sparsely-sampled input images was 20.03 dB when reconstructed directly using IFFT. The network trained for 300 epochs with noisy targets reached an average PSNR of 31.74 dB on the validation data, and the network trained with clean targets reached 31.77 dB. Here the training with clean targets is similar to prior art (Wang et al., 2016; Lee et al., 2017). Training took 13 hours on an NVIDIA Tesla P100 GPU. Figure 9(b, c) shows an example of reconstruction results between convolutional networks trained with noisy and clean targets, respectively. In terms of PSNR, our results quite closely match those reported in recent work.



(a) Input 18.93 dB (b) Noisy trg. 29.77 dB (c) Clean trg. 29.81 dB (d) Reference

Figure 9. MRI reconstruction example. (a) Input image with only 10% of spectrum samples retained and scaled by $1/p$. (b) Reconstruction by a network trained with noisy target images similar to the input image. (c) Same as previous, but training done with clean target images similar to the reference image. (d) Original, uncorrupted image. PSNR values refer to the images shown here, see text for averages over the entire validation set.

4. Discussion

We have shown that simple statistical arguments lead to new capabilities in learned signal recovery using deep neural networks; it is possible to recover signals under complex corruptions *without observing clean signals*, without an explicit statistical characterization of the noise or other corruption, at performance levels equal or close to using clean target data. That clean data is not necessary for denoising is not a new observation: indeed, consider, for instance, the classic BM3D algorithm (Dabov et al., 2007) that draws on self-similar patches within a single noisy image. We show that the previously-demonstrated high restoration performance of deep neural networks can likewise be achieved entirely without clean data, all based on the same general-purpose deep convolutional model. This points the way to significant benefits in many applications by removing the need for potentially strenuous collection of clean data.

AmbientGAN (Ashish Bora, 2018) trains generative adversarial networks (Goodfellow et al., 2014) using corrupted observations. In contrast to our approach, AmbientGAN needs an explicit forward model of the corruption. We find combining ideas along both paths intriguing.

[4] Our simplified example deviates from practical MRI in the sense that we do not sample the spectra along 1D trajectories. However, we believe that designing pulse sequences that lead to similar pseudo-random sampling characteristics is straightforward.

[5] http://brain-development.org/ixi-dataset → T1 images.

Acknowledgments

Bill Dally, David Luebke, Aaron Lefohn for discussions and supporting the research; NVIDIA Research staff for suggestions and discussion; Runa Lober and Gunter Sprenger for synthetic off-line training data; Jacopo Pantaleoni for the interactive renderer used in on-line training; Samuli Vuorinen for initial photography test data; Koos Zevenhoven for discussions on MRI; Peyman Milanfar for helpful comments.

References

Ashish Bora, Eric Price, Alexandros G. Dimakis. AmbientGAN: Generative models from lossy measurements. *ICLR*, 2018.

Cerdá-Company, Xim, Párraga, C. Alejandro, and Otazu, Xavier. Which tone-mapping operator is the best? A comparative study of perceptual quality. *CoRR*, abs/1601.04450, 2016.

Chaitanya, Chakravarty R. Alla, Kaplanyan, Anton S., Schied, Christoph, Salvi, Marco, Lefohn, Aaron, Nowrouzezahrai, Derek, and Aila, Timo. Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. Graph.*, 36 (4):98:1–98:12, 2017.

Dabov, K., Foi, A., Katkovnik, V., and Egiazarian, K. Image denoising by sparse 3-D transform-domain collaborative filtering. *IEEE Trans. Image Process.*, 16(8):2080–2095, 2007.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative Adversarial Networks. In *NIPS*, 2014.

Hasinoff, Sam, Sharlet, Dillon, Geiss, Ryan, Adams, Andrew, Barron, Jonathan T., Kainz, Florian, Chen, Jiawen, and Levoy, Marc. Burst photography for high dynamic range and low-light imaging on mobile cameras. *ACM Trans. Graph.*, 35(6):192:1–192:12, 2016.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

Huber, Peter J. Robust estimation of a location parameter. *Ann. Math. Statist.*, 35(1):73–101, 1964.

Iizuka, Satoshi, Simo-Serra, Edgar, and Ishikawa, Hiroshi. Globally and locally consistent image completion. *ACM Trans. Graph.*, 36(4):107:1–107:14, 2017.

Isola, Phillip, Zhu, Jun-Yan, Zhou, Tinghui, and Efros, Alexei A. Image-to-image translation with conditional adversarial networks. In *Proc. CVPR 2017*, 2017.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Ledig, Christian, Theis, Lucas, Huszar, Ferenc, Caballero, Jose, Aitken, Andrew P., Tejani, Alykhan, Totz, Johannes, Wang, Zehan, and Shi, Wenzhe. Photo-realistic single image super-resolution using a generative adversarial network. In *Proc. CVPR*, pp. 105–114, 2017.

Lee, D., Yoo, J., and Ye, J. C. Deep residual learning for compressed sensing MRI. In *Proc. IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*, pp. 15–18, 2017.

Lustig, Michael, Donoho, David L., Santos, Juan M., and Pauly, John M. Compressed sensing MRI. In *IEEE Signal Processing Magazine*, volume 25, pp. 72–82, 2008.

Maas, Andrew L, Hannun, Awni Y, and Ng, Andrew. Rectifier nonlinearities improve neural network acoustic models. In *Proc. International Conference on Machine Learning (ICML)*, volume 30, 2013.

Mao, Xiao-Jiao, Shen, Chunhua, and Yang, Yu-Bin. Image restoration using convolutional auto-encoders with symmetric skip connections. In *Proc. NIPS*, 2016.

Martin, D., Fowlkes, C., Tal, D., and Malik, J. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. ICCV*, volume 2, pp. 416–423, 2001.

Mäkitalo, Markku and Foi, Alessandro. Optimal inversion of the Anscombe transformation in low-count Poisson image denoising. *IEEE Trans. Image Process.*, 20(1): 99–109, 2011.

Reinhard, Erik, Stark, Michael, Shirley, Peter, and Ferwerda, James. Photographic tone reproduction for digital images. *ACM Trans. Graph.*, 21(3):267–276, 2002.

Ronneberger, Olaf, Fischer, Philipp, and Brox, Thomas. U-net: Convolutional networks for biomedical image segmentation. *MICCAI*, 9351:234–241, 2015.

Rousselle, Fabrice, Knaus, Claude, and Zwicker, Matthias. Adaptive sampling and reconstruction using greedy error minimization. *ACM Trans. Graph.*, 30(6):159:1–159:12, 2011.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

Ulyanov, Dmitry, Vedaldi, Andrea, and Lempitsky, Victor S. Deep image prior. *CoRR*, abs/1711.10925, 2017.

Veach, Eric and Guibas, Leonidas J. Optimally combining sampling techniques for Monte Carlo rendering. In *Proc. ACM SIGGRAPH 95*, pp. 419–428, 1995.

Wang, S., Su, Z., Ying, L., Peng, X., Zhu, S., Liang, F., Feng, D., and Liang, D. Accelerating magnetic resonance imaging via deep learning. In *Proc. IEEE 13th International Symposium on Biomedical Imaging (ISBI)*, pp. 514–517, 2016.

Zeyde, R., Elad, M., and Protter, M. On single image scale-up using sparse-representations. In *Proc. Curves and Surfaces: 7th International Conference*, pp. 711–730, 2010.

Zhang, Richard, Isola, Phillip, and Efros, Alexei A. Colorful image colorization. In *Proc. ECCV*, pp. 649–666, 2016.

A. Appendix

A.1. Network architecture

Table 2 shows the structure of the U-network (Ronneberger et al., 2015) used in all of our tests, with the exception of the first test in Section 3.1 that used the "RED30" network (Mao et al., 2016). For all basic noise and text removal experiments with RGB images, the number of input and output channels were $n = m = 3$. For Monte Carlo denoising we had $n = 9, m = 3$, i.e., input contained RGB pixel color, RGB albedo, and a 3D normal vector per pixel. The MRI reconstruction was done with monochrome images ($n = m = 1$). Input images were represented in range $[-0.5, 0.5]$.

A.2. Training parameters

The network weights were initialized following He et al. (2015). No batch normalization, dropout or other regularization techniques were used. Training was done using ADAM (Kingma & Ba, 2015) with parameter values $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 10^{-8}$.

Learning rate was kept at a constant value during training except for a brief rampdown period at where it was smoothly brought to zero. Learning rate of 0.001 was used for all experiments except Monte Carlo denoising, where 0.0003 was found to provide better stability. Minibatch size of 4 was used in all experiments.

A.3. Finite corrupted data in L_2 minimization

Let us compute the expected error in L_2 norm minimization task when corrupted targets $\{\hat{y}_i\}_{i=1}^N$ are used in place of the clean targets $\{y_i\}_{i=1}^N$, with N a finite number. Let y_i be arbitrary random variables, such that $\mathbb{E}\{\hat{y}_i\} = y_i$. As usual, the point of least deviation is found at the respective mean. The expected squared difference between these means across realizations of the noise is then:

$$
\begin{aligned}
&\mathbb{E}_{\hat{y}} \left[\frac{1}{N} \sum_i y_i - \frac{1}{N} \sum_i \hat{y}_i \right]^2 \\
=&\frac{1}{N^2} \left[\mathbb{E}_{\hat{y}} (\sum_i y_i)^2 - 2\mathbb{E}_{\hat{y}} \left[(\sum_i y_i)(\sum_i \hat{y}_i) \right] + \mathbb{E}_{\hat{y}} (\sum_i \hat{y}_i)^2 \right] \\
=&\frac{1}{N^2} \mathrm{Var}(\sum_i \hat{y}_i) \\
=&\frac{1}{N} \left[\frac{1}{N} \sum_i \sum_j \mathrm{Cov}(\hat{y}_i, \hat{y}_j) \right]
\end{aligned}
\tag{8}
$$

In the intermediate steps, we have used $\mathbb{E}_{\hat{y}}(\sum_i \hat{y}_i) = \sum_i y_i$ and basic properties of (co)variance. If the corruptions are mutually uncorrelated, the last row simplifies to

$$
\frac{1}{N} \left[\frac{1}{N} \sum_i \mathrm{Var}(y_i) \right] \tag{9}
$$

In either case, the variance of the estimate is the average (co)variance of the corruptions, divided by the number of samples N. Therefore, the error approaches zero as the number of samples grows. The estimate is unbiased in the

NAME	N_{out}	FUNCTION
INPUT	n	
ENC_CONV0	48	Convolution 3×3
ENC_CONV1	48	Convolution 3×3
POOL1	48	Maxpool 2×2
ENC_CONV2	48	Convolution 3×3
POOL2	48	Maxpool 2×2
ENC_CONV3	48	Convolution 3×3
POOL3	48	Maxpool 2×2
ENC_CONV4	48	Convolution 3×3
POOL4	48	Maxpool 2×2
ENC_CONV5	48	Convolution 3×3
POOL5	48	Maxpool 2×2
ENC_CONV6	48	Convolution 3×3
UPSAMPLE5	48	Upsample 2×2
CONCAT5	96	Concatenate output of POOL4
DEC_CONV5A	96	Convolution 3×3
DEC_CONV5B	96	Convolution 3×3
UPSAMPLE4	96	Upsample 2×2
CONCAT4	144	Concatenate output of POOL3
DEC_CONV4A	96	Convolution 3×3
DEC_CONV4B	96	Convolution 3×3
UPSAMPLE3	96	Upsample 2×2
CONCAT3	144	Concatenate output of POOL2
DEC_CONV3A	96	Convolution 3×3
DEC_CONV3B	96	Convolution 3×3
UPSAMPLE2	96	Upsample 2×2
CONCAT2	144	Concatenate output of POOL1
DEC_CONV2A	96	Convolution 3×3
DEC_CONV2B	96	Convolution 3×3
UPSAMPLE1	96	Upsample 2×2
CONCAT1	$96+n$	Concatenate INPUT
DEC_CONV1A	64	Convolution 3×3
DEC_CONV1B	32	Convolution 3×3
DEV_CONV1C	m	Convolution 3×3, linear act.

Table 2. Network architecture used in our experiments. N_{out} denotes the number of output feature maps for each layer. Number of network input channels n and output channels m depend on the experiment. All convolutions use padding mode "same", and except for the last layer are followed by leaky ReLU activation function (Maas et al., 2013) with $\alpha = 0.1$. Other layers have linear activation. Upsampling is nearest-neighbor.

sense that it is correct on expectation, even with a finite amount of data.

The above derivation assumes scalar target variables. When $\hat{y}_i$ are images, N is to be taken as the total number of scalars in the images, i.e., #images $\times$ #pixels/image $\times$ #color channels.

A.4. Mode seeking and the "L_0" norm

Interestingly, while the "L_0 norm" could intuitively be expected to converge to an exact mode, i.e. a local maximum of the probability density function of the data, theoretical analysis reveals that it recovers a slightly different point. While an actual mode is a zero-crossing of the *derivative* of the PDF, the L_0 norm minimization recovers a zero-crossing of its *Hilbert transform* instead. We have verified this behavior in a variety of numerical experiments, and, in practice, we find that the estimate is typically close to the true mode. This can be explained by the fact that the Hilbert transform approximates differentiation (with a sign flip): the latter is a multiplication by $i\omega$ in the Fourier domain, whereas the Hilbert transform is a multiplication by $-i\,\mathrm{sgn}(\omega)$.

For a continuous data density $q(x)$, the norm minimization task for L_p amounts to finding a point x^* that has a minimal expected p-norm distance (suitably normalized, and omitting the pth root) from points $y \sim q(y)$:

$$\begin{aligned} x^* &= \operatorname*{argmin}_x \mathbb{E}_{y\sim q}\{\frac{1}{p}|x-y|^p\} \\ &= \operatorname*{argmin}_x \int \frac{1}{p}|x-y|^p q(y)\,\mathrm{d}y \end{aligned} \tag{10}$$

Following the typical procedure, the minimizer is found at a root of the derivative of the expression under argmin:

$$\begin{aligned} 0 &= \frac{\partial}{\partial x}\int \frac{1}{p}|x-y|^p q(y)\,\mathrm{d}y \\ &= \int \mathrm{sgn}(x-y)|x-y|^{p-1} q(y)\,\mathrm{d}y \end{aligned} \tag{11}$$

This equality holds also when we take $\lim_{p\to 0}$. The usual results for L_2 and L_1 norms can readily be derived from this form. For the L_0 case, we take $p = 0$ and obtain

$$\begin{aligned} 0 &= \int \mathrm{sgn}(x-y)|x-y|^{-1} q(y)\,\mathrm{d}y \\ &= \int \frac{1}{x-y} q(y)\,\mathrm{d}y. \end{aligned} \tag{12}$$

The right hand side is the formula for the Hilbert transform of $q(x)$, up to a constant multiplier.